گزارش‌کار پروژه ی الگوریتم

طراحی الگوریتم‌ها

استاد راهنما: دکتر پیمان ادیبی

اعضای گروه:

مهرآذین مرزوق - 4003613055

پوریا طلائی - 4003623024

بهار 1402

فهرست مطالب

طراحی

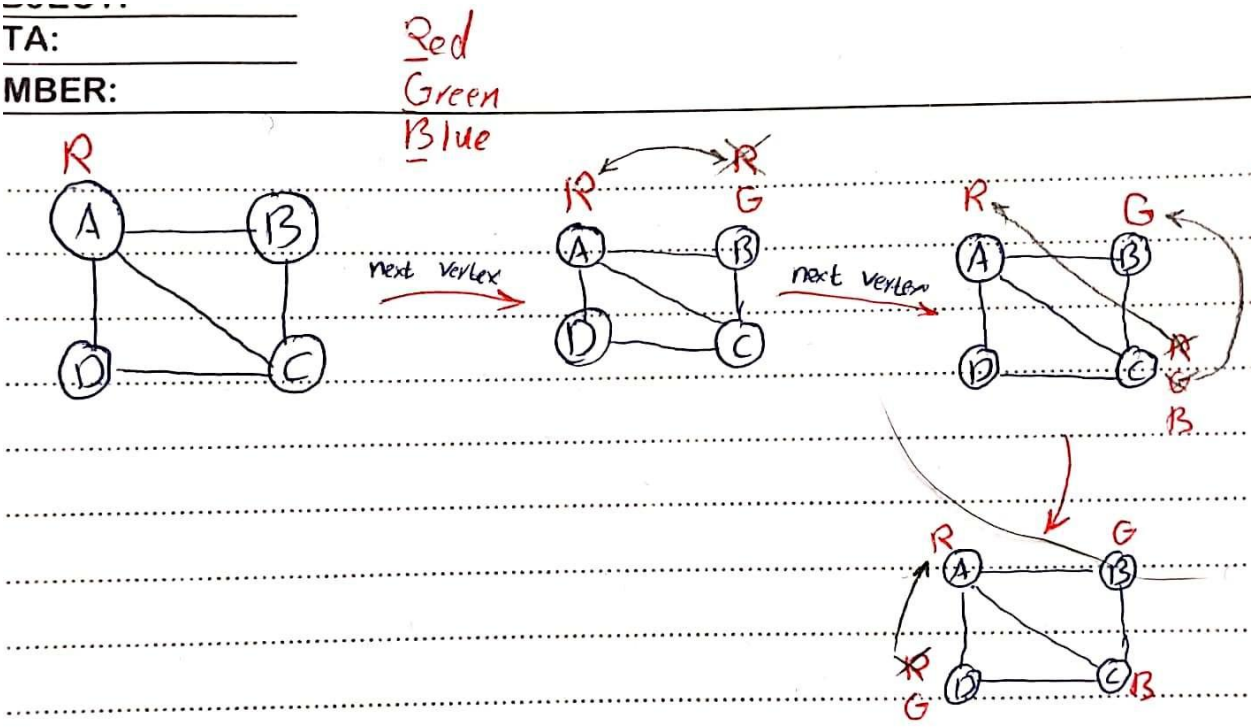
طراحی فرش‌های جدید

خواسته مسئله

خواسته‌ی مسئله از ما این است که کمترین تعداد رنگ مورد نیاز برای طراحی یک فرش را بدست آوریم و در ادامه یک مثال از این طراحی را به کاربر نشان دهیم.

روش حل مسئله

برای حل این مسئله از الگوریتم رنگ‌کردن گراف بهره گرفتیم. نحوه‌ی عملکرد آن در تصویر زیر نشان داده شده است.



در این متد الگوریتم بالا را پیاده‌سازی کردیم. به طوری که با دریافت گراف، تعداد رنگ‌ها، اندیس i ، آرایه‌ای برای ست کردن رنگها و تعداد رئوس مقدار بولینی به ما باز می‌گرداند.

در تابع isSafe چک می‌شود که رنگ دو خانه کنار هم یکسان نباشد.

اگر مقدار true بازگشت یعنی رنگ‌آمیزی درستی صورت گرفته‌است.

هنگامی که اولین true بازگردد m کمترین مقدار قابل قبول می‌باشد.(جواب مسئله)

وظیفه‌ی تابع findMin همین است.

پیچیدگی زمانی آن o(m^v*v) و مقدار حافظه‌ای که اشغال می‌کند برابر است با n^2

```
static boolean graphColoring(int[][] graph, int m,
                             int i, Integer[] color, int V) {
    // if current index reached end
    if (i == V + 1) {

        // if coloring is safe
        if (isSafe(graph, color, V)) {
            return true;
        }
        return false;
    }

    // Assign each color from 1 to m
    for (int j = 1; j <= m; j++) {
        color[i] = j;
        if(isSafe(graph,color,i)){
            if(graphColoring(graph,m, i+1,color,V))
                return true;
        }



    }
    return false;
}
```

```
static boolean isSafe(int[][] graph, Integer[] color, int i)
{
    // check for every edge
    for(int j = 1; j < i; j++){
        if(graph[j][i] == 1 && color[i].equals(color[j]))
            return false;
    }

    return true;
}
```

```java
private DesignAnswer findMinM(Graph graph){
    int V = graph.getN();
    Integer[] color = new Integer[V+1];
    DesignAnswer ans = new DesignAnswer(color, m: 0);

    int[][] g = graph.getAdj();


    int m;
    for(m = 2; m < V+1; m++) {
        for(int i = 0; i < V+1; i++){
            color[i] = 0;
        }
        if(graphColoring(g, m, i: 1, color,V)){
            ans = new DesignAnswer(color,m);
            break;
        }
    }

    return ans;
}
```

ورودی و خروجی

```
Main ×
"C:\Program Files\Java\jdk-20\bin\java.exe"
WELCOME
1: Design
2: Sale
0: Exit
1
NUMBER OF VERTICES:
3
WHAT VERTICES ARE CONNECTED TOGETHER?
0 0: END
1 2
2 3
0 0
colors=
    area1: 1
    area2: 2
    area3: 1

m= 2
```

فروش

جست‌وجو بر اساس طرح نقشه

خواسته مسئله

در این مسئله از ما خواسته شده تا سه تا از شبیه‌ترین فرش‌های کارخانه به فرش واردشده توسط کاربر را نشان دهیم.

روش حل

برای حل این مسئله از الگوریتم هم‌ترازی دنباله‌ها بهره گرفتیم.

```
public String[] searchByMap(Graph200 graph) throws FileNotFoundException {
    String input = graphToStr(graph);
    Similarity[] similarities = new Similarity[5];


    for(int i = 0; i < 5; i++){
        String carpet = readFile(i);
        int[][] arr = new int[carpet.length()][input.length()];
        similarities[i] = new Similarity(sequenceAlignment(carpet,input, i: 0, j: 0,arr),carpet);
    }

    sort(similarities, low: 0, high: 4);
    String[] ans = new String[3];
    for(int i = 0; i < 3; i++){
        ans[i] = strToGraph(similarities[i].graph);
    }
    return ans;
```

```java
private int sequenceAlignment(String carpet,String input, int i, int j, int[][] arr){
    int ans = 0;
    int opt;
    int penalty;
    if(i == carpet.length()) {
        ans = 2 * (input.length() - j - 1);
        arr[i][j] = ans;
    }
    else if (j == input.length()) {
        ans = 2 * (input.length() - j - 1);
        arr[i][j] = ans;
    }
    else{
        if(carpet.charAt(i) == input.charAt(j))
            penalty = 0;
        else
            penalty = 1;
        opt = Math.min(arr[i+1][j]+2,arr[i][j+1]+2);
        ans = Math.min(arr[i+1][j+1]+penalty,opt);
    }
    return ans;
}
```

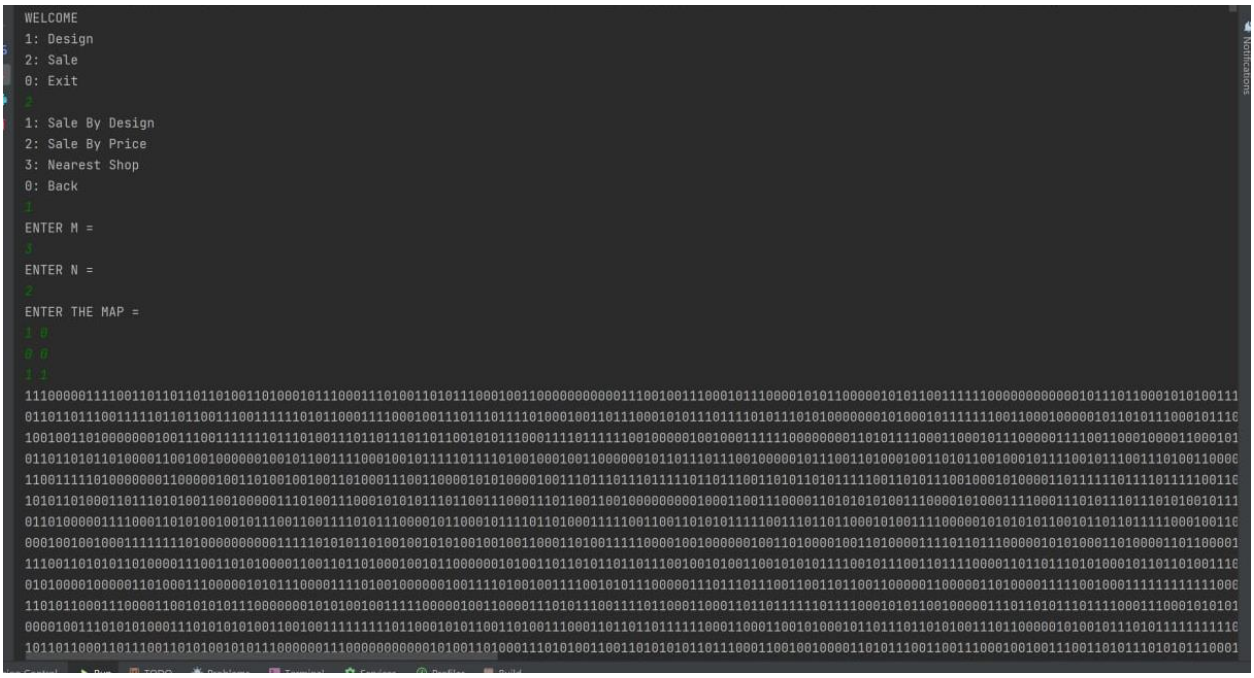
همانگونه که مشاهده می‌کنید متد searchByMapدر این کلاس وجود دارد که یک شیء از کلاس Graph200را دریافت می‌کند و یک آرایه از رشته‌ها را برمی‌گرداند. هدف این متد پیدا کردن سه فرش شبیه به ورودی است. برای این منظور، این متد از الگوریتم sequenceAlignment برای محاسبه‌ی شباهت بین ورودی و هر یک از فرش‌ها استفاده می‌کند. سپس، آرایه‌ی شباهت‌ها را مرتب می‌کند و سه فرش با بیشترین شباهت را به عنوان خروجی برمی‌گرداند .مرتب‌سازی با روش quick sort صورت می‌گیرد.

متد sequenceAlignment چهار ورودی دریافت می‌کند: یک رشته به نامcarpet ، یک رشته به نام input، دو عدد صحیح به نام i و j و یک آرایه دو بعدی از اعداد صحیح به نام .arr هدف این متد محاسبه شباهت بین دو رشته است. الگوریتم استفاده شده در این متد برای محاسبه شباهت بین دو رشته استفاده می‌شود و با استفاده از یک جدول دو بعدی از اعداد صحیح که در آن هر خانه نشان دهنده شباهت بین دو زیررشته است، عملکرد می‌کند. در هر مرحله، الگوریتم یک خانه از جدول را پر می‌کند. برای پر کردن هر خانه، الگوریتم از سه خانه‌ی قبلی در جدول استفاده می‌کند و یک هزینه (penalty)را به آن اضافه می‌کند. سپس، کمینه این سه مقدار را به عنوان مقدار خانه‌ی جدید در نظر می‌گیرد. در صورتی که دو حرف در همان موقعیت در دو رشته یکسان باشند، هزینه برابر با صفر است و در غیر این صورت، هزینه برابر با یک است. در صورتی که یکی از رشته‌ها به پایان برسد، هزینه برابر با دو برابر تعداد حروف باقی‌مانده در رشته دیگر است. در نهایت، مقدار شباهت بین دو رشته در خانه آخر جدول قرار داده می‌شود و به عنوان خروجی برگردانده می‌شود.

```
private int partition(Similarity arr[], int low, int high)
{
    Similarity pivot = arr[high];
    int i = (low-1); // index of smaller element
    for (int j=low; j<high; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j].similarity <= pivot.similarity)
        {
            i++;

            // swap arr[i] and arr[j]
            Similarity temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // swap arr[i+1] and arr[high] (or pivot)
    Similarity temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;

    return i+1;
}
```

```
private void sort(Similarity arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is
          now at right place */
        int pi = partition(arr, low, high);

        // Recursively sort elements before
        // partition and after partition
        sort(arr, low, high: pi-1);
        sort(arr, low: pi+1, high);
    }
}
```

ورودی و خروجی



خرید بر اساس میزان پول

خواسته مسئله

مسئله از ما می‌خواهد که بیشترین تعداد فرشی که می‌توان با مقدار مشخصی پول خرید را به دست آوریم.

روش حل

در این مسئله از الگوریتم مرتب‌سازی سریع بهره گرفتیم.

```
private ArrayList<CarpetPrice> findByBudget(CarpetPrice[] carpetPrice, int budget){
    ArrayList<CarpetPrice> ans = new ArrayList<>();
    int money = 0;
    for(int i = 0; i < carpetPrice.length; i++){
        money += carpetPrice[i].price;
        if(money <= budget)
            ans.add(carpetPrice[i]);
        else
            break;
    }
    return ans;
```

```
public String searchByPrice(int budget) throws FileNotFoundException {
    CarpetPrice[] carpetPrices = new CarpetPrice[5];
    for(int i = 0; i < 5; i++){
        carpetPrices[i] = readFile(i);
    }

    sort(carpetPrices, low: 0, high: 4);
    ArrayList<CarpetPrice> found = findByBudget(carpetPrices,budget);
    String ans = arrayListToStr(found);

    return ans;
```

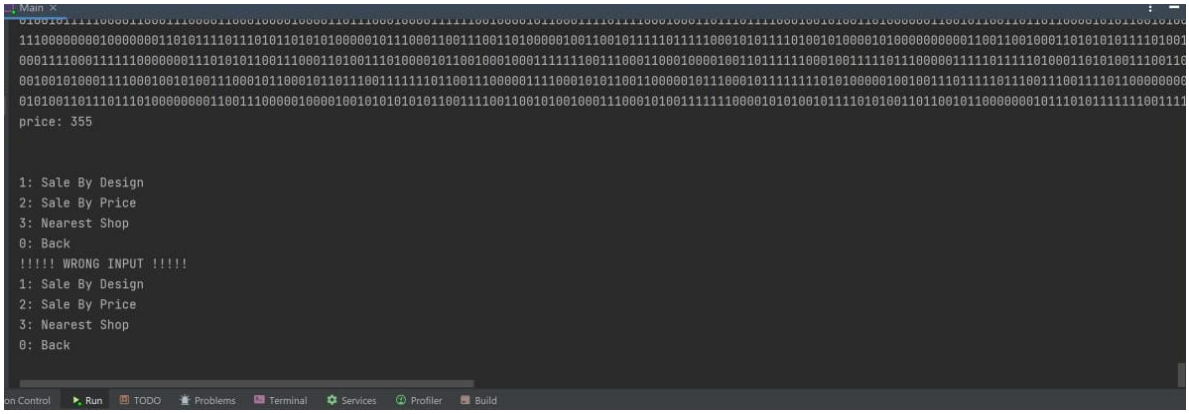
به طوری که برای هر فرش پارامتری برای قیمت در نظر گرفتیم. سپس آن‌ها را بر اساس قیمت مرتب‌سازی می‌کنیم (از طریق quick sort که در مسئله‌ی قبلی نیز عنوان شد)

حال بیشترین تعداد فرشی که می‌توانیم بخریم برابر است با ارزان‌قیمت‌ترین فرش‌ها به شرطی که مجموعشان از مقدار پول کاربر بیشتر نشود.

پیچیدگی زمانی آن برابر است با o(logn)

و مقدار حافظه‌ی مورد استفاده برابر است با

ورودی و خروجی



مسیریابی به نزدیکترین فروشگاه کارخانه

خواسته مسئله

در این مسئله از ما خواسته شده تا شعبه‌ای که به مختصات ورودی کاربر نزدیک‌تر است را به دست آوریم و مسیری منتهی به آن را نمایش دهیم.

روش حل

الگوریتمی که برای حل این مسئله استفاده کردیم الگوریتم فلوید می‌باشد.

```
public class SaleByLocation {
    1 usage
    public SaleByLocation() throws FileNotFoundException {
        makeCity();
        floyd();
    }
```

در ابتدا با استفاده از متد makeCity() در کلاس GraphCity ماتریس استاتیک ۱۶ در ۱۶ ای ساخته می‌شود که matrix[i][j] بیانگر طول خیابان بین چهارراه i و j می‌باشد.

```
private void makeCity() throws FileNotFoundException {
    File file = new File( pathname: "City Map.txt");
    Scanner scanner = new Scanner(file);
    int[][] adj = new int[16][16];
    for(int i = 0; i < 16; i++){
        for(int j = 0; j < 16; j++){
            String ch = scanner.next();
            if(i != j && ch.equals("0")){
                adj[i][j] = 1000;
                continue;
            }
            adj[i][j] = Integer.parseInt(ch);
        }
    }
    GraphCity.setAdj(adj);
    scanner.close();
    file = new File( pathname: "map.txt");
    scanner = new Scanner(file);
    String[][] street = new String[16][16];
    for(int i = 0; i < 16; i++){
        for(int j = 0; j < 16; j++){
            String ch = scanner.next();
            street[i][j] = ch;
        }
    }
    GraphCity.setSteets(street);
}
```

متد Floyd() در کلاس GraphCity ماتریس ۱۶ در ۱۶ ای می‌سازد که matrix[i][j] بیانگر بزرگترین اندیس از یک راس واسطه روی کوتاه‌ترین مسیر از راس i به راس j می باشد. در صورتی که مسیری بین این دو راس وجود نداشته باشد، این خانه حاوی عدد ۰ می‌باشد.

```java
private void floyd(){
    int[][] D = GraphCity.getAdj();
    int[][] P = new int[16][16];

    for(int k = 1; k<16; k++){
        for(int i = 1; i < 16; i++){
            for (int j = 1; j < 16; j++) {
                if(D[i][k] + D[k][j] < D[i][j]){
                    P[i][j] = k;
                    D[i][j] = D[i][k] + D[k][j];
                }
            }
        }
    }

    GraphCity.setP(P);
    GraphCity.setMinDistance(D);
}
```

متد path() به صورت بازگشتی، مسیر بین راس i و j را مشخص می‌کند.

```java
3 usages
private ArrayList<Integer> path(int x, int y){
    int[][] P = GraphCity.getP();
    ArrayList<Integer> ans = new ArrayList<>();

    if(GraphCity.getMinDistance()[x][y] != 0){
        if(x != 0 && y != 0 && P[x][y] != 0) {
            path(x, P[x][y]);
            ans.add(P[x][y]);
        }
        if(x != 0 && y != 0)
            path(P[x][y],y);
    }
    ans.add(y);
    return ans;
}
```

در متد saleByLocation() با دریافت لوکیشن کاربر، کوتاه‌ترین مسیرها بین تمام شعبه‌ها با یکدیگر مقایسه شده و در نهایت کوتاه‌ترین این مسیرها نمایش داده می‌شود.

```
public String saleByLocation(int x, int y) throws FileNotFoundException {
    int p = (x-1)*5 + y;
    ArrayList<Integer> path = new ArrayList<>();
    for (int i = 0; i < 16; i++){
        path.add(0);
    }

    for(int i = 0; i < 16; i++){
        if(i == 2 || i == 4 || i == 5 || i == 8 || i == 11 || i == 15){
            ArrayList<Integer> tmpPath = new ArrayList<>();
            tmpPath.add(p);
            tmpPath.addAll(path(p,i));
            if(tmpPath.size() < path.size())
                path = tmpPath;

        }
    }

    String address = "";
    String[][] s = GraphCity.getSteets();
    for(int i = 0; i < path.size(); i++){
        if( i != 0)
            address += GraphCity.getSteets()[path.get(i-1)][path.get(i)] + " Street - ";
        address += path.get(i) + " Intersection - ";


    }
    return address;
}
```

پیچیدگی زمانی این الگوریتم برابر با پیچیدگی زمانی path() می‌باشد که به‌علت بازگشتی بودن الگوریتم $\theta(logn)$ می‌باشد.

ورودی و خروجی

```
WELCOME
1: Design
2: Sale
0: Exit
2
1: Sale By Design
2: Sale By Price
3: Nearest Shop
0: Back
3
ENTER YOUR X =
3
ENTER YOUR Y =
3
13 Intersection - t Street - 8 Intersection
```

منابع

https://www.geeksforgeeks.org/m-coloring-problem/

Foundations of Algorithms - Richard Neapolitan - Book